МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра САПР

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Компьютерная графика»

Тема: Формирование различных поверхностей с использованием ее пространственного разворота и ортогонального проецирования на плоскость при ее визуализации (выводе на экран дисплея)

Студенты гр. 1302 _____ Марзаева В.И.

_____ Новиков Г.В.

_____ Романова О.В.

Преподаватель _____ Колев Г.Ю.

Санкт-Петербург

2024

Цель работы

Сформировать билинейную поверхность на основе произвольного задания ее четырех угловых точек. Обеспечить ее поворот относительно осей X и Y.

Теоретическая часть программы

Билинейные поверхности – простейшие трехмерные поверхности. Они задаются на ограниченном участке с заданием в пространстве 4-х угловых точек поверхности. Уравнение билинейчатой поверхности представляется как:

$$\overline{Q}(u,w) = \overline{P}_{00}(1-u)(1-w) + \overline{P}_{01}(1-u)w + \overline{P}_{10}u(1-w) + \overline{P}_{11}uw$$

$$0 \le u \le 1$$

$$0 \le w \le 1$$

Если u=0; w=0, то попадаем в точку $\overline{P}_{00} = \overline{Q}(u,w)$

Если u=1; w=0, то попадаем в точку $\overline{P}_{10} = \overline{Q}(u,w)$

Если u=1; w=1, то попадаем в точку $\overline{P}_{11} = \overline{Q}(u,w)$

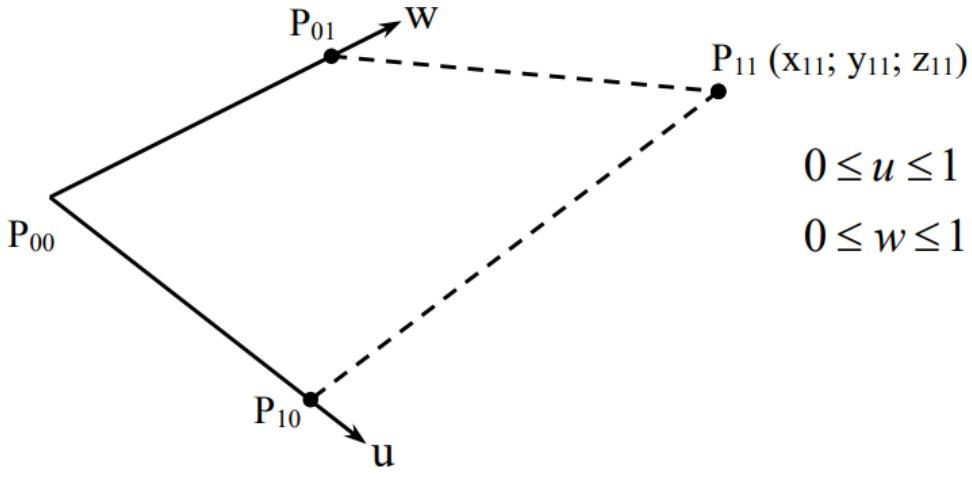

Рис. 1

Пример работы программы

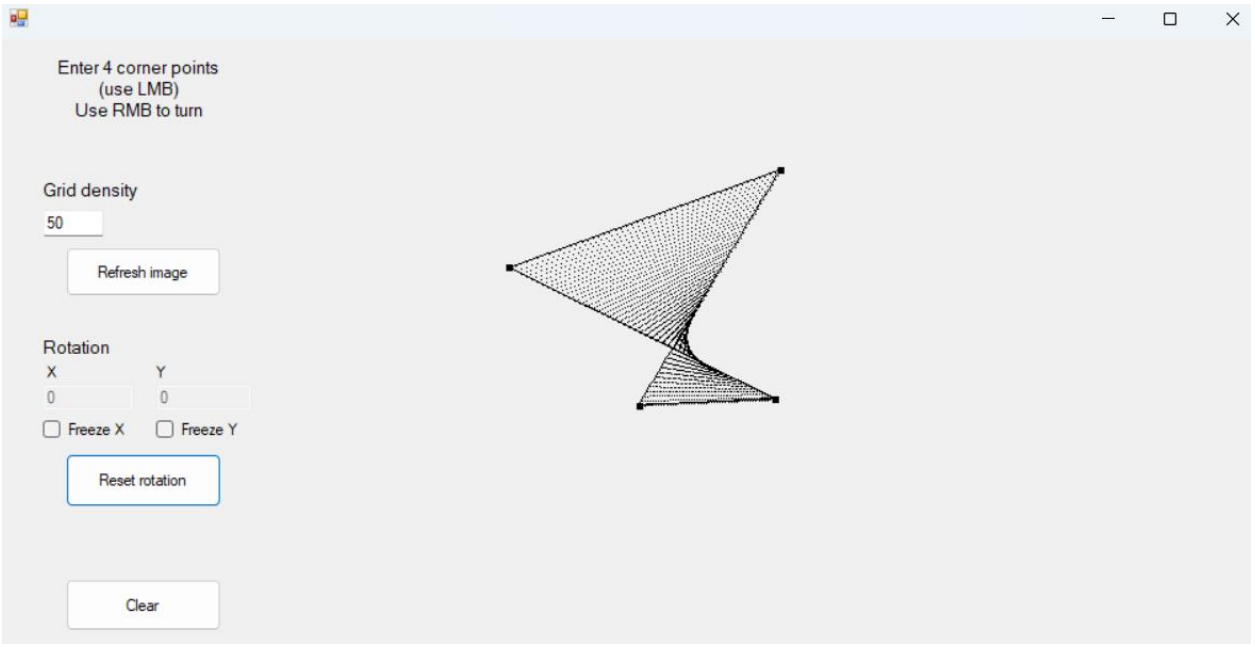

Рис. 3 – Пример работы программы

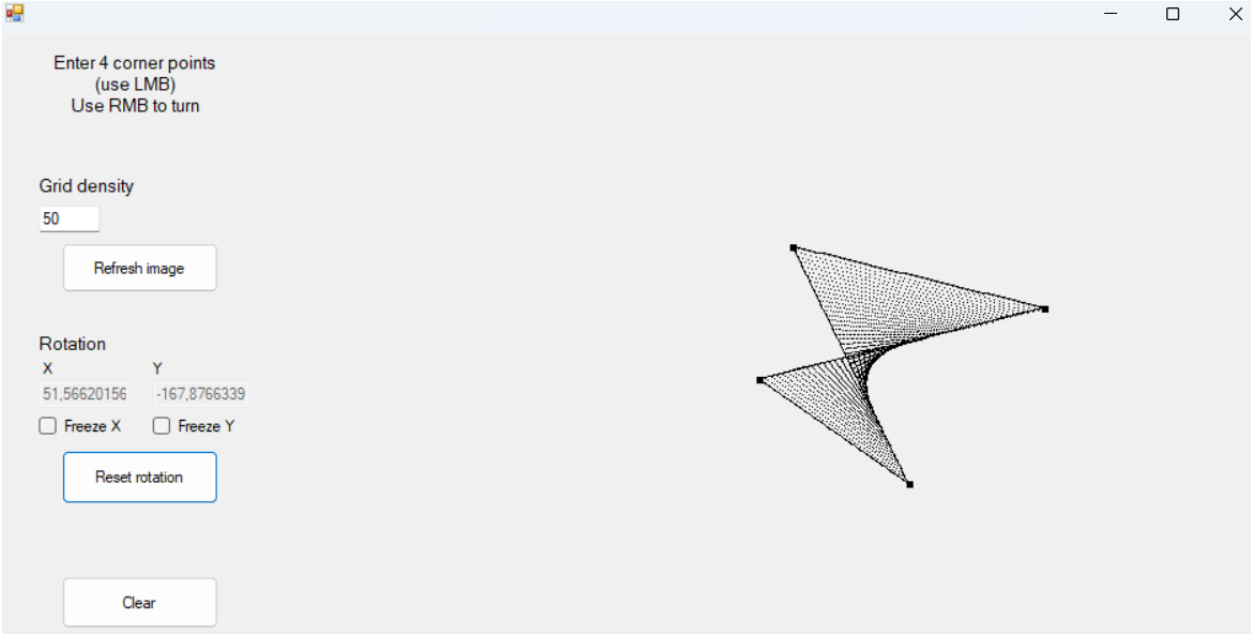

Рис. 4 – Пример работы программы

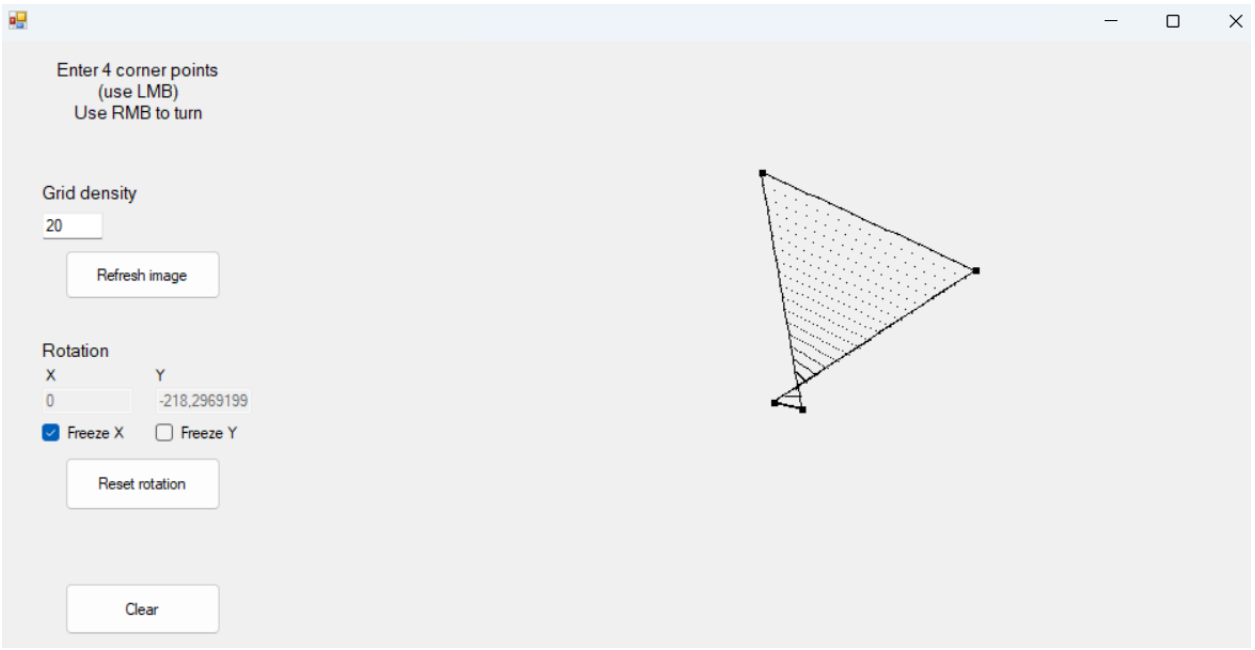

Рис. 5 – Пример работы программы

Код программы

```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Drawing.Imaging;
using System.Linq;
using System.Windows.Forms;

using MathNet.Numerics.LinearAlgebra;

namespace lab_3
{
   public partial class Form1 : Form
   {
      private Vector<double>[] corners = new Vector<double>[4];
      private int cornerIndex = 0;
      private Vector<double> center = Vector<double>.Build.DenseOfArray(new[] {0d, 0d, 0d});

      private bool rightMousePressed = false;
      private Point mouseDownPoint = new Point(0, 0);
      private Vector<double> rotation = Vector<double>.Build.Dense(2);

      private bool cursorHidden = false;
      private Point cursorFixPosition;

      private int usualPointSize = 1;
      private int cornerPointSize = 5;
      private double sensitivityX = 0.01;
      private double sensitivityY = 0.01;

      public int GridDensity
      {
         get { return int.TryParse(textBox1.Text, out int density) ? density : 0; }
      }
```

```
public bool FreezeX
{
   get { return checkBox1.Checked; }
}

public bool FreezeY
{
   get { return checkBox2.Checked; }
}

public Form1()
{
   InitializeComponent();
   textBox1.Text = "50";
   center[0] = pictureBox1.Width / 2;
   center[1] = pictureBox1.Height / 2;
}

private void PictureBox1_Paint(object sender, PaintEventArgs e)
{
   if (corners.All(p => p != null) && GridDensity != 0)
   {
      label2.Text = "";
      DrawBilinearSurface(corners, e.Graphics, GridDensity);
   }
   else
   {
      DrawCornerPoints(e.Graphics);
   }

   textBox5.Text = (rotation[0] * 180 / Math.PI).ToString();
   textBox6.Text = (rotation[1] * 180 / Math.PI).ToString();
   DisplayMessage();
}

private void PictureBox1_MouseClick(object sender, MouseEventArgs e)
{
   if (e.Button == MouseButtons.Left && cornerIndex < corners.Length)
   {
      Vector<double> point = Vector<double>.Build.DenseOfArray(new[] { e.Location.X, e.Location.Y, 0d });
      point -= center;
      RotatePointY(ref point, -rotation[1]);
      RotatePointX(ref point, -rotation[0]);
      corners[cornerIndex] = point;
      cornerIndex++;

      pictureBox1.Refresh();
   }
}

private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
{
   if (rightMousePressed)
   {
      if (!FreezeX) rotation[0] += (e.Y - mouseDownPoint.Y) * sensitivityX;
      if (!FreezeY) rotation[1] += (e.X - mouseDownPoint.X) * sensitivityY;
      pictureBox1.Refresh();
      Cursor.Position = cursorFixPosition;
   }
}

private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
{
   if (e.Button == MouseButtons.Right)
   {
```

```
            rightMousePressed = true;
            mouseDownPoint.X = e.X;
            mouseDownPoint.Y = e.Y;
            if (!cursorHidden)
            {
                Cursor.Hide();
                cursorHidden = true;
                cursorFixPosition = Cursor.Position;
            }
        }
    }

    private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
    {
        if (e.Button == MouseButtons.Right)
        {
            rightMousePressed = false;
            if (cursorHidden)
            {
                Cursor.Show();
                cursorHidden = false;
            }
        }
    }

    private void button1_Click(object sender, EventArgs e)
    {
        rotation.Clear();
        pictureBox1.Refresh();
    }

    private void button2_Click(object sender, EventArgs e)
    {
        pictureBox1.Refresh();
    }

    private void Button3_Click(object sender, EventArgs e)
    {
        ClearForm();
    }

    private void DisplayMessage()
    {
        if (GridDensity <= 1)
        {
            label2.ForeColor = Color.Red;
            label2.Text = "Invalid grid density";
        }
        else
        {
            label2.ForeColor = Color.Black;
            label2.Text = "Enter 4 corner points\n(use left mouse button)\nUse RMB to turn";
        }
    }

    private void DrawCornerPoints(Graphics g)
    {
        for (int i = 0; i < corners.Length; i++)
        {
            Vector<double> corner = corners[i];
            if (corner != null)
            {
                Vector<double> point = corner;
                RotatePointX(ref point, rotation[0]);
                RotatePointY(ref point, rotation[1]);
```

```
            DrawPoint(point, cornerPointSize, g);
        }
    }
}

private void DrawPoint(Vector<double> point, int fatness, Graphics g)
{
    point += center;
    g.FillRectangle(Brushes.Black, (float)point[0] - fatness / 2, (float)point[1] - fatness / 2, fatness, fatness);
}

private void DrawLine(Vector<double> point1, Vector<double> point2, Graphics g)
{
    point1 += center;
    point2 += center;
    if (GridDensity > 1)
    {
        g.DrawLine(Pens.Black, new Point((int)point1[0], (int)point1[1]), new Point((int)point2[0], (int)point2[1]));
    }
}

private void DrawBilinearSurface(Vector<double>[] corners, Graphics g, int grid_density)
{
    double du = 1.0 / (grid_density / 1 - 1);
    double dw = 1.0 / (grid_density / 1 - 1);

    Vector<double> prevUpBorderPoint = CalculatePointOnSurface(corners, 0, 0);
    Vector<double> prevDownBorderPoint = CalculatePointOnSurface(corners, 1, 1);
    Vector<double> prevRightBorderPoint = CalculatePointOnSurface(corners, 0, 1);
    Vector<double> prevLeftBorderPoint = CalculatePointOnSurface(corners, 1, 0);

    int pointSize = usualPointSize;
    for (int i = 0; i < grid_density; i++)
    {
        double u = i * du;
        for (int j = 0; j < grid_density; j++)
        {
            double w = j * dw;
            Vector<double> point = CalculatePointOnSurface(corners, u, w);

            bool isCorner = (i == 0 && (j == 0 || j + 1 == grid_density));
            isCorner = isCorner || (i + 1 == grid_density && (j == 0 || j + 1 == grid_density));

            pointSize = isCorner ? cornerPointSize : usualPointSize;
            DrawPoint(point, pointSize, g);

            bool isBorder = (i == 0 || j == 0 || i + 1 == grid_density || j + 1 == grid_density);
            if (isBorder)
            {
                if (i == 0)
                {
                    DrawLine(point, prevUpBorderPoint, g);
                    prevUpBorderPoint = point;
                }
                if (i + 1 == grid_density)
                {
                    DrawLine(point, prevDownBorderPoint, g);
                    prevDownBorderPoint = point;
                }
                if (j == 0)
                {
                    DrawLine(point, prevLeftBorderPoint, g);
                    prevLeftBorderPoint = point;
                }
                if (j + 1 == grid_density)
```

```
                {
                    DrawLine(point, prevRightBorderPoint, g);
                    prevRightBorderPoint = point;
                }
            }
        }
    }
}

private Vector<double> CalculatePointOnSurface(Vector<double>[] corners, double u, double w)
{
    Vector<double> point = Vector<double>.Build.Dense(3);
    if (corners.Any(p => p == null))
    {
        return point;
    }

    for (int i = 0; i < 3; i++)
    {
        point[i] = corners[0][i] * (1 - u) * (1 - w) + corners[1][i] * (1 - u) * w + corners[2][i] * u * (1 - w) + corners[3][i] * u *
w;
    }

    RotatePointX(ref point, rotation[0]);
    RotatePointY(ref point, rotation[1]);

    return point;
}

private void RotatePointX(ref Vector<double> v, double angle)
{
    Matrix<double> matrix = GetXRotationMatrix(angle);
    v = matrix * v;
}

private void RotatePointY(ref Vector<double> v, double angle)
{
    Matrix<double> matrix = GetYRotationMatrix(angle);
    v = matrix * v;
}

private Matrix<double> GetXRotationMatrix(double angle)
{
    double cos = Math.Cos(angle);
    double sin = Math.Sin(angle);

    double[,] rotX = {
        { 1, 0, 0},
        { 0, cos, -sin},
        { 0, sin, cos}
    };

    return Matrix<double>.Build.DenseOfArray(rotX);
}

private Matrix<double> GetYRotationMatrix(double angle)
{
    double cos = Math.Cos(angle);
    double sin = Math.Sin(angle);

    double[,] rotY = {
        { cos, 0, sin},
        { 0, 1, 0},
        { -sin, 0, cos}
    };
```

```
            return Matrix<double>.Build.DenseOfArray(rotY);
        }

        private void ClearForm()
        {
            Array.Clear(corners, 0, corners.Length);
            cornerIndex = 0;
            rotation.Clear();
            pictureBox1.Refresh();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            pictureBox1.Anchor = AnchorStyles.Top | AnchorStyles.Bottom | AnchorStyles.Left | AnchorStyles.Right;

        }
    }
}
```

Выводы

В данной работе с помощью Windows Forms на C# была реализована программа, которая строит билинейную поверхность по заданным пользователем четырем угловым точкам. Также был реализован поворот относительно осей X и Y, с использованием возможности «замораживания» x и y.